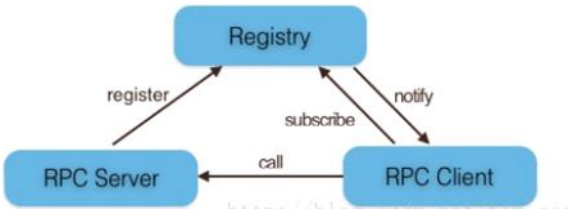
远程调用

2019年3月16日 13:48

-
- 远程调用

1. 远程过程调用RPC
   1) 一种广义的远程调用技术框架，程序可向网络中的另一台计算机上的程序请求服务
   2) RPC 的主要目的是为组件提供一种相互通信的方式，使这些组件之间能够相互发出请求并传递这些请求的结果
   3) 在 RPC 中，发出请求的程序是客户程序，而提供服务的程序是服务器
   4) 广泛用于支持分布式应用程序的技术
2. RPC实现方案
   1) 从0开始：基于SOCKET网络通信和序列化的编程实现
      i. 用接口实现服务功能
      ii. 编写javabean类，将调用相关信息（类名或接口名、方法名、方法参数类型数组、方法参数值数组和返回结果）封装在里面，把该类对象序列化通过网络传给服务器
      iii. 注册中心：用map保存服务的接口
      iv. 网络传输：javabean对象remotecall通过tcp协议（客户端用Socket对象和对象输出流）传到服务器，（服务端用Server类对象.register(功能类/接口名，功能类对象).exportService()，将serverSocket.accept()）将服务器执行结果remotecall对象原路返回
   2) RMI（java）
   3) 基于第三方的框架（Netty通信框架， gRPC、 Dubbo，Thrift等）
   4) SOA架构（WebService等）
   5) 新发展：RESTful等
3. RPC框架原理
   1) Server: 暴露服务的服务提供方。
   2) Client: 调用远程服务的服务消费方。
   3) Registry: 服务注册与发现的注册中心。
   4) 

   5) 实现的关键技术问题
      i. 服务发现
      ii. 通讯
      iii. 数据封装与序列化

4. 调用流程
    1) 服务消费方（client）调用以本地调用方式调用服务；
    2) client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；
    3) client stub找到服务地址，并将消息发送到服务端；
    4) server stub收到消息后进行解码；
    5) server stub根据解码结果调用本地的服务；
    6) 本地服务执行并将结果返回给server stub；
    7) server stub将返回结果打包成消息并发送至消费方；
    8) client stub接收到消息，并进行解码；
    9) 服务消费方得到最终结果。
    10) 目标：通过服务注册、发现与调用机制，将2-8步封装起来
5. Remote Method Invocation框架RMI
    1) jdk1.1中实现的，用代理负责Socket通信的类，在客户端生成存根stub，在服务器端生成skeleton，右远程对象的skeleton调用远程对象的方法
        i. RMI定位远程对象的机制是命名服务机制，被整合进了JNDI中（Java Naming and Directory Interface）
        ii. 远程服务类需实现的接口：java.rmi.Remote，java.rmi.server.UnicastRemoteObject
    2) 服务注册中心的生成：调用LocateRegistry.createRegistry(int port)在本机指定端口打开注册中心
        i. 注册：把服务对象和jndi资源名绑定
        ii. 发现：从jndi中检索资源名对应的服务对象
    3) Naming/Context/Registry类常用方法
        i. bind(String name,Object obj): 如果name已与其他对象绑定，抛出NameAlreadyBoundException
        ii. rebind(String name,Object obj):不会抛出NameAlreadyBoundException,obj指定的对象覆盖原来已绑定的对象
        iii. Object lookup(String name) : 返回name所绑定的对象（完整URL为rmi://主机名:端口号1099/xx；本地可以忽略前缀）
        iv. void unbind(String name):注销对象
    4) 优势：简单，常用于内部局域网的java平台
    5) 劣势：局限于java，注册中心功能少，效率和性能一般，安全性不够，灵活性扩展性低
6. RMI开发
    1) 基本流程
        i. 创建远程接口：直接或间接继承java.rmi.Remote接口（接口中不包含任何抽象方法，仅做标记表明该接口的实现类对象能被远程主机所调用）
        ii. 创建远程类：实现远程接口，将远程类的对象导出为远程接口
        iii. 创建服务器程序：负责在rmiregistry注册中心内注册远程对象。

iv. 创建客户程序：负责定位远程对象，并且调用远程对象的方法。

2) 创建远程接口
   i. 必须条件：
      1) 直接或间接继承java.rmi.Remote接口
      2) 接口中的所有方法声明抛出java.rmi.RemoteException或父异常
   ii. 直接继承：
      1) public interface XXXService extends Remote {
         public XXX method1() throws RemoteException;
         public XXX method2() throws RemoteException;
        }
   iii. 间接继承：
      1) public interface A {
         public XXX method1() throws Exception;
         public XXX method2() throws IOException; }
      2) public interface XXXService extends A,Remote { }
      3) 优点：不需修改服务接口本身，就能支持RMI，这也使系统中RMI细节被隐藏

3) 创建远程类
   i. 需继承java.rmi.server.UnicastRemoteObject类，并且构造方法必须声明抛出RemoteException，UnicastRemoteObject构造方法的内容：
      1) protected UnicastRemoteObject() throws RemoteException
         { this(0); }
      2) protected UnicastRemoteObject(int port) throws RemoteException
         { this.port = port; exportObject((Remote) this, port); }
      3) public static Remote exportObject(Remote obj, int port) throws RemoteException
   ii. 若不能继承该类，也可在构造方法中调用静态方法exportObject()并声明抛出RemoteException
      1) class XXX extends OtherClass implements RemoteXXX{
         public XXX() throws RemoteException{
           ……..;
           UnicastRemoteObject.exportObject(this,0);
         }
        }
   iii. 也可在服务器程序中直接调用UnicastRemoteObject.exportObject(),把一个实现了远程接口的类的对象导出为远程对象

4) 创建服务器端程序
   i. 生成远程类对象
   ii. 在指定端口打开服务注册中心

iii. 将该对象绑定到注册中心

5) 创建客户端程序

i. 通过JNDI在注册中心找到想访问的远程对象（的存根）

ii. 调用存根对象的方法